

A Decidable Timeout based Extension of Propositional Linear Temporal Logic

Janardan Misra

EMCSS India Pvt. Ltd., Bangalore 560048, India.

Email: janmishra@gmail.com

Suman Roy

SETLABS, Infosys Tech. Ltd., #44 Electronic City,
Bangalore 560100, India.

Email: suman_roy@infosys.com

December 20, 2010

Abstract

We develop a timeout based extension of propositional linear temporal logic (which we call TLTL) to specify timing properties of timeout based models of real time systems. TLTL formulas explicitly refer to a running global clock together with static timing variables as well as a dynamic variable abstracting the timeout behavior. We extend LTL with the capability to express timeout constraints. From the expressiveness view point, TLTL is not comparable with important known clock based real-time logics including TPTL, XCTL, and MTL, i.e., TLTL can specify certain properties, which cannot be specified in these logics (also vice-versa). We define a corresponding timeout tableau for satisfiability checking of the TLTL formulas. Also a model checking algorithm over timeout Kripke structure is presented. Further we prove that the validity checking for such an extended logic remains PSPACE-complete even in the presence of timeout constraints and infinite state models. Under discrete time semantics, with bounded timeout increments, the model-checking problem that if a TLTL-formula holds in a timeout Kripke structure is also PSPACE complete. We further prove that when TLTL is interpreted over discrete time, it can be embedded in the monadic second order logic with time, and when TLTL is interpreted over dense time without the condition of non-zenoness, the resulting logic becomes Σ_1^1 -complete.

Keywords: Timeout systems, Real time logics, Model checking, Timing properties, Timeout constraints, Tableau satisfiability, Undecidability

1 Introduction

Real-time systems are an important class of mission critical systems, which have been well studied for their design, implementation, and performance [OD08]. Designing faithful models for real-time systems essentially requires representing different kinds of timing behavior e.g., relative delays and timing constraints.

In a timeout based design framework for real-time systems, timing requirements are modeled by defining the execution of an action in terms of an expiration of a delay, often represented as a timeout (or timer). Traditionally, timeouts have been used in real-time system designs for handling various timing scenarios including (forced) expiration of a waiting state. Dutertre and Sorea [DS04] used timeout based modeling to formally verify safety properties of the real-time systems with discrete dynamics. A timeout model contains a finite set of timeouts and a variable x which keeps track of the current (global) time. Timeouts define the time points when discrete transitions are enabled in the future. In practice, a typical real-time system may contain n concurrently active processes. Each process is associated with one timeout which denotes the future point of time when the next discrete transition for the corresponding process will occur. Transitions in this model are classified into two types - time progress transitions and discrete transitions. In a time progress transition, the time variable x is advanced to the minimum valued timeout(s). A discrete transition occurs when x is equal to the minimum valued timeout(s). If there are more than one processes, which have their timeouts equal to the minimum value, then some of them are randomly selected and corresponding discrete transitions take place with the values of the corresponding timeouts are set in the future.

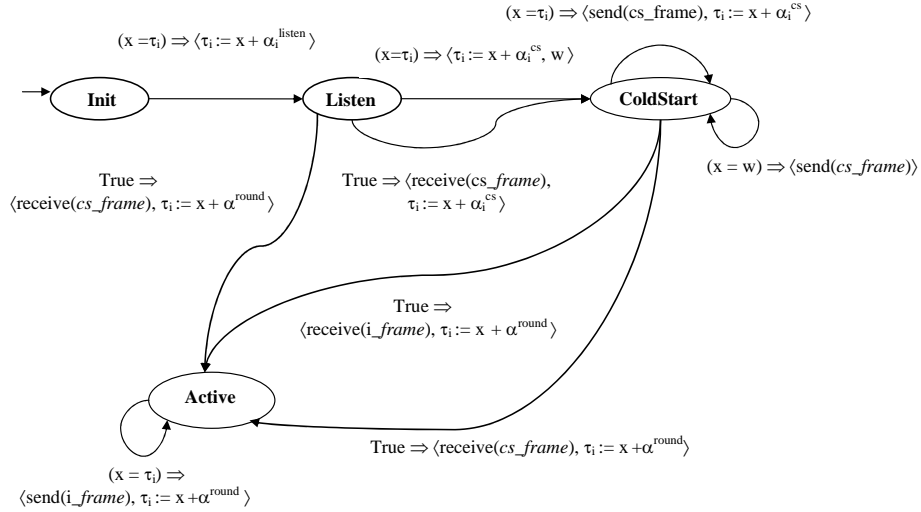


Figure 1: State transition diagram of TTA startup algorithm at i^{th} node. Edges are labeled as: $\text{guard} \Rightarrow \langle [\text{send/receive}], \text{timeout update}, [\text{record_time_var}] \rangle$, where (optional) record_time_var records the time when a transition occurs on the edge.

Startup algorithm for Time-triggered Architecture (TTA) is an example of a system where timeouts are explicitly used in the design. TTA start-up algorithm executes on a logical bus meant for safety critical application in both automotive and aerospace industries. In a normal operation, N nodes share a TTA bus using a TDMA schedule. The state-machine of the startup algorithm executed on the nodes is shown in Figure 1. Each node $i \in [1, N]$ has a local timeout τ_i . Timeout increments in various states are defined in terms of timeout increment parameters: $\alpha_i^{listen} = (2N + i - 1)\lambda$, $\alpha_i^{cs} = (N + i - 1)\lambda$, and $\alpha_i^{round} = N\lambda$, where

λ refers to the (fixed) duration of each slot in a TDMA schedule. When a node is powered-on, it transits from *init* state to *listen* state and listens for the duration α_i^{listen} to determine if there is a synchronous set of nodes communicating on the medium. Similarly a node in *coldstart* state waits for reception of frames until clock x reaches the value of its timeout. If it receives such a frame, it enters the *active* state, else it broadcasts another frame, loops into the coldstart state, and waits for another α_i^{cs} time units. For a brief description of the TTA startup algorithm, the reader is referred to [DS04]. For a detailed exposition to startup protocols, we refer the reader to [SP02].

Denoting the minimum of all timeout values in any state by y , a timeout event in a state can be characterized by constraint $x = y$. Also the following properties might be of interest¹.

- In each state, either a timeout occurs or it is set in the future:

$$\Box((x = y) \vee (x < y))$$

- If a node i comes to the *listen* state (characterized by p_{lis}) at time $x = t_0$, it will move to the *coldstart* state (characterized by p_{cs}) in time no later than $x = t_0 + \alpha_i^{listen}$:

$$\forall t_0. \Box(p_{lis} \wedge (x = t_0) \Rightarrow \Diamond(p_{cs} \wedge (x \leq t_0 + \alpha_i^{listen}))).$$

Clearly ordinary linear temporal logic (LTL) [Pnu77, LP85] would not allow expressing such properties. One needs to extend LTL with the capability to express timeout constraints. Even the popular real-time extensions of LTL, e.g., TPTL [AH94] cannot be used in a straightforward manner to express these constraints. For example, in order to use TPTL for model checking the TTA start-up model discussed above, the model would have to be redesigned using explicit clock based frameworks e.g., timed automata [AD94]. These clock based models in turn need to explicitly simulate the timeout semantics as discussed before. Also, as we shall discuss in Section 4.2, certain liveness properties on global timeout events cannot be expressed using TPTL.

The primary objective of this work is to develop a real-time extension of LTL that can handle timeout constraints and possesses an efficient model checking algorithm as well. Over the past decade, there has been a sustained effort to increase the expressive power of temporal logic, which is a popular mechanism for specifying and verifying temporal properties of reactive and real-time systems. As we discuss further in Section 4, several attempts have been made to incorporate time explicitly into LTL, and to interpret the resulting logics over models that associate a time with every state. Examples of such logics are RTTL [Ost89], XCTL [HLP90], TPTL [AH94], MTL [Koy90] etc. Quite a few verification tools have been developed based on these logics, e.g., DT-SPIN [BD98], RT-SPIN [TC96], UPPAAL [BDL04]. Since these tools adopt clock-based modeling approaches they can be used to formalize timeout systems only by first converting the timeout models into clock-based models (e.g., timed automata with clocks). On the other hand, the infinite bounded model checker of SAL (Symbolic Analysis Laboratory) [dMOR⁺04] can model timeout

¹For the formal semantic interpretation of these formulas see Section 2.

systems, however supports only LTL model checking and demands considerable manual efforts while defining supporting lemmas and abstractions during the model checking process. In order to alleviate such problems timeout based modeling was earlier formalized by the authors in [SMR07] in terms of predicate transition diagrams and the current work deals with defining the corresponding specification logic and model checking procedure.

The remainder of the paper is organized as follows: The logic TLTL is introduced in the Section 2. In Section 3, we introduce a monadic second order (MSO) theory of timeout state sequences and prove that TLTL when interpreted over discrete time can be embedded in it. In Section 4, we compare TLTL with other real-time extensions of LTL including XCTL, TPTL, and MTL. In Section 5, we describe a tableau based decision procedure for the validity (and satisfiability) checking of TLTL formulas followed by its complexity analysis. Model checking of TLTL formulas over timeout Kripke structures is discussed in Section 6 with associated complexity analysis. In Section 7 we prove an undecidability result under dense time interpretation without time progress constraint. We conclude with a discussion on the directions for future work in Section 8.

2 The Logic TLTL

In this section we will define the syntax and semantics of *Timeout based Propositional Linear Temporal Logic*, TLTL.

2.1 Syntax of TLTL

The basic vocabulary of TLTL consists of a finite set \mathcal{P} of propositions **true**, **false**, p, q, \dots , a finite set T of (global) static timing variables t_1, t_2, \dots . In addition, we allow a dynamic variable x which represents the clock and a dummy variable y^2 . Assume $\Delta = \{<, =, >\}$, and let \sim range over Δ . We use $\mathbb{R}^{\geq 0}$ to denote the set of non negative real numbers, and \mathbb{N} to denote the set of non negative integers.

- The set of atomic formulas (A_f) consists of propositions in \mathcal{P} and atomic constraints of the form $x < y, x = y, x < u, x = u$, and $x > u$ where $u ::= t + c \mid c$, t ranges over T and $c \in \mathbb{N}$ is a constant. We will refer $x < u, x = u$ and $x > u$ as **static constraints** and $x < y$ and $x = y$ as **dynamic constraints**.
- (unquantified) Formulas are built using the following grammar

$$\phi ::= a_f \mid \phi \vee \phi \mid \neg \phi \mid \bigcirc \phi \mid \phi \mathcal{U} \phi$$

where a_f ranges over A_f .

²Variable y is essentially a place holder for minimum of the timeouts in a timeout program, which will be introduced in Section 6.1. This abstraction is adopted primarily because, according to the behavior of a timeout system as discussed in Section 1, a discrete transition in a state may occur only when the current time is equal to the minimum valued timeout. For convenience, y will also be referred sometimes as ‘minimum of the timeouts’.

- Finally, a quantified formula is built using universal quantification over timing variables at the outermost level as:

$$\psi ::= \forall t_1 t_2 \dots t_k. \phi,$$

where $T_\psi = \{t_1, t_2, \dots, t_k\} \subseteq T$ is the set of timing variables appearing in the (unquantified) formula ϕ .

- The additional operators $\wedge, \Rightarrow, \Leftrightarrow$ and modal operators \Diamond, \Box are introduced as abbreviations, $p \Rightarrow q \equiv \neg p \vee q$, $\Diamond \phi \equiv \mathbf{true} \mathcal{U} \phi$, $\Box \phi \equiv \neg(\Diamond \neg \phi)$.

2.2 Semantics of TLTL

We consider the following point-wise or (timeout) event based semantics for TLTL. Towards defining a model for a TLTL formula consider a sequence of states of the form

$$\sigma : s_0 s_1 \dots,$$

such that each s_i gives a boolean interpretation (**true**, **false**) to the propositions, and non negative real valued interpretation to the timing variables in T , to the clock variable x , and the variable y .

In a state s_i , let us assume that $s_i(x)$ denotes the value of the clock variable x , $s_i(y)$ the value of variable y , and $s_i(t_j)$ the value of timing variable $t_j \in T$. It is further required that

(m_1) *Monotonicity*: Clock x and variable y do not decrease:

$$\forall i : s_i(x) \leq s_{i+1}(x) \text{ and } s_i(y) \leq s_{i+1}(y)$$

(m_2) *Time Progress*: To ensure effective time progress in the model a *divergence* condition³, which says that time eventually increases, is required:

$$\forall \delta \in \mathbb{R}^{\geq 0} : \exists i \text{ such that } s_i(x) > \delta$$

(m_3) *State Transition*: Upon a change of state either timeout variables stay constant or some of them increase, that is, for each i :

- if the clock in state s_i is less than the minimum of the timeouts, i.e. y , clock advances to this value in the next state s_{i+1} :

$$[(s_i(x) < s_i(y)) \Rightarrow (s_{i+1}(y) = s_i(y)) \wedge (s_{i+1}(x) = s_i(y))]$$

- else, if the clock in state s_i is equal to the value of y , in the next state s_{i+1} , y advances in the future:

$$[(s_i(x) = s_i(y)) \Rightarrow (s_{i+1}(y) > s_i(y)) \wedge (s_{i+1}(x) = s_i(x))]$$

As a consequence we have for each i , $s_i(x) \leq s_i(y)$, that is, timeouts are always set in the future.

³This is also known in the literature as ‘non-zenoness’ or ‘finite-variability’ condition.

(m_4) *Initiality*: For the initial state s_0 the following hold: either, $s_0(x) = s_0(y) = 0$ (when $x = y$ holds in s_0) or $0 = s_0(x) < s_0(y)$ (when $x < y$ holds in s_0).

(m_5) *Constant Interpretation for Static Variables*: All the states are required to assign the same interpretation to the static timing variables, that is, for a given formula ψ ,

$$\forall t_j \in T_\psi : s_i(t_j) = s_0(t_j), \text{ for each } i.$$

Thus a model for a TLTL formula may contain infinitely many different states with different values of the clock and timeout variables. Boolean and modal operators are given the usual interpretation. We mention only atomic formulas in A_f which are interpreted in a state as follows.

$$\begin{array}{lll} s_i \models p & \text{iff} & s_i(p) = \mathbf{true} \\ s_i \models x \smile t_j + c & \text{iff} & s_i(x) \smile s_i(t_j) + c \\ s_i \models x \smile' y & \text{iff} & s_i(x) \smile' s_i(y) \end{array}$$

Finally, we define $\sigma \models \psi$ iff $s_0 \models \phi$ for any interpretation of the static timing variables appearing in ϕ given in state s_0 . The formula ψ is satisfiable (valid) if $\sigma \models \psi$ for some (all) sequence(s) σ .

For example, consider **time bounded response property**, which specifies that “event q is always followed by event p within 5 time units”. It can be expressed by a TLTL formula

$$\forall t_0. \Box(p \wedge (x = t_0) \Rightarrow \Diamond(q \wedge (x \leq t_0 + 5))) \quad (1)$$

We can also consider a variant of this as a **bounded timeout response property** stating that “timeout event q is always followed by timeout event p within 5 time units”, which would be expressed by a TLTL formula

$$\forall t_0. \Box(p \wedge (x = y) \wedge (x = t_0) \Rightarrow \Diamond(q \wedge (x = y) \wedge (x \leq t_0 + 5))) \quad (2)$$

A quantified formula ψ is termed as *closed* if all the timing variables appearing in it are bounded by a universal quantifier. In the rest of the discussion we will only consider closed quantified formulas. Also we will follow usual notational convention [Ost89, PH88, HLP90] of implicit universal quantification and would often drop the outermost universal quantification over global static timing variables in T . For example, the time bounded response property, specified by the TLTL formula (1) would actually be written as

$$\Box(p \wedge (x = t_0) \Rightarrow \Diamond(q \wedge (x \leq t_0 + 5))) \quad (\phi_{BR_{TLTL}})$$

A formula of the form $x \leq z$ ($z := u|y$) is an abbreviation for $(x < z) \vee (x = z)$. Similarly $x \geq u$ abbreviates $(x > u) \vee (x = u)$. Note that $x > y$ is not a valid formula in TLTL.

3 An Embedding of TLTL in MSO

In this section we explore the relationship of TLTL with monadic second order logic (MSO) with time. Towards that we consider an interpretation of MSO in integer time structure. Subsequently we provide a straightforward meaning preserving translation between TLTL and monadic logic.

3.1 Monadic Second Order Theory of Timeout State Sequences

Next we briefly recall the theory of timed state sequences \mathcal{L}_2^T as introduced in [AH93] and extend it slightly. This is defined by adding a linearly ordered time domain $(TIME, <)$ with the theory of state sequences, S1S [Buc60], through a monotonically non decreasing function $f : \mathbb{N} \mapsto TIME$ that associates a time with every state in the sequence. Thus a timed state sequence is a pair (σ', f) consisting of an infinite sequence of states $\sigma' = s'_0 s'_1 \dots$ and function f .

Let us additionally consider monotonically non decreasing function $g : \mathbb{N} \mapsto TIME$ representing the minimum of the timeouts in a state. This defines a *timeout state sequence* as a triple (σ', f, g) .

Let \mathcal{L}_2^T be a second-order language with two sorts, a state sort and a time sort as considered in [AH93]. The vocabulary of the congruence free sub language of \mathcal{L}_2^T consists of:

- The sets Var_1 and Var_2 for state sort. Set $Var_1 = \{i, j, \dots\}$ consists of individual (first order) variables and the set $Var_2 = \{p, q, \dots\}$ contains (second-order) set or predicate variables.
- The binary predicate symbol $<$ over the state and time sort;
- The unary function symbol f from the state sort into the time sort;
- The quantification over individual variables in Var_1 and over predicate variables in Var_2 .

Let \mathcal{L}_2^{TO} be the language which in addition to \mathcal{L}_2^T also contains:

- The unary function symbols g from the state sort into the time sort;
- The set of additional unary function symbols $Var_2^t = \{\mathbf{t}_1, \mathbf{t}_2, \dots\}$ from the state sort into the time sort;

We consider only those formulas which do not contain any free individual variables. Further, we restrict our attention to structures that choose the set of natural numbers \mathbb{N} as domain for both sorts with usual linear order $<$ on them. Given a formula ϕ of \mathcal{L}_2^{TO} with the free predicate variables $p_1, \dots, p_n \in Var_2$ and free function symbols $\mathbf{T}_\psi = \{\mathbf{t}_1, \dots, \mathbf{t}_k\} \subseteq Var_2^t$, an interpretation I for ϕ specifies the sets $p_1^I, \dots, p_n^I \subseteq \mathbb{N}$, monotonically non decreasing functions $f^I : \mathbb{N} \mapsto \mathbb{N}$ and $g^I : \mathbb{N} \mapsto \mathbb{N}$, $\mathbf{t}_1^I : \mathbb{N} \mapsto \mathbb{N}, \dots, \mathbf{t}_k^I : \mathbb{N} \mapsto \mathbb{N}$. The satisfaction relation \models is defined in a standard fashion.

Every interpretation I for ϕ implicitly defines a timeout state sequence (σ', f, g) : Let σ' be the infinite sequence of states $s'_0 s'_1 \dots$, where $s'_i \in 2^{\{p_1, \dots, p_n\}} \times \mathbb{N}^k$ such that $(p_j, (n_1, \dots, n_k)) \in s'_i \Leftrightarrow i \in p_j^I$ and $\forall 1 \leq j \leq k. \mathbf{t}_j^I(i) = n_j$. Also let $f = f^I$ and $g = g^I$ for notational convenience.

\mathcal{L}_2^{TO} -formulas define properties of timeout state sequences. For example, a bounded timeout response property discussed earlier (ref. Eq. (2)), and can be defined by a formula

$$\forall i. (p(i) \wedge (f(i) = g(i)) \Rightarrow \exists j \geq i. (q(j) \wedge (f(j) = g(j)) \wedge (f(j) \leq f(i) + 5)))$$

$(\phi_{BR_{\mathcal{L}_T}})$

An \mathcal{L}_2^{TO} -formula ϕ is satisfiable (valid) if it is satisfied by some (every) timeout state sequence. The (second-order) theory of timeout state sequences is the set of all valid formulas of \mathcal{L}_2^{TO} . The following result is an immediate adaptation of the decidability result from [AH93]:

Fact 1 (Decidability) *The validity problem for the language \mathcal{L}_2^{TO} is decidable.*

3.2 TLTL as a fragment of \mathcal{L}_2^{TO}

Now we provide a meaning preserving compositional translation of TLTL formulas into \mathcal{L}_2^{TO} . Every TLTL-formula $\psi := \forall t_1 \dots t_k. \phi$ can be translated into \mathcal{L}_2^{TO} , while preserving the set of models of ψ . The translation will use $\mathbf{T}_\psi = \{\mathbf{t}_1, \dots, \mathbf{t}_k\}$ to capture the static timing variables in $T_\psi = \{t_1, \dots, t_k\}$, and a free individual variable $i \in \text{Var}_1$ acting as a state counter. For every proposition p of TPTL, we use a corresponding unary predicate $p(i)$ of state sort. We translate a TLTL-formula ψ to the \mathcal{L}_2^{TO} -formula

$$\text{Tr}(\psi) = \forall i. \forall \mathbf{t}_j \in \mathbf{T}_\psi. [\Lambda_{m_2} \wedge \Lambda_{m_3} \wedge \Lambda_{m_4} \wedge \Lambda_{m_5} \wedge \text{Tr}_0(\phi)]$$

where semantic constraints of TLTL as defined in Section 2.2 are encoded by $\Lambda_{m_2} \dots \Lambda_{m_5}$:

$$\begin{aligned} \Lambda_{m_2} : & \quad \forall l \in \mathbb{N}. \exists m \in \mathbb{N}. f(m) > l \\ \Lambda_{m_3} : & \quad [f(i) < g(i) \Rightarrow (g(i+1) = g(i)) \wedge (f(i+1) = g(i))] \\ & \quad \vee [f(i) = g(i) \Rightarrow (g(i+1) > g(i)) \wedge (f(i+1) = f(i))] \\ & \quad \vee \neg[f(i) > g(i)] \\ \Lambda_{m_4} : & \quad [(f(0) = 0) \wedge (g(0) = 0)] \vee [(f(0) \geq 0) \wedge (f(0) < g(0))] \\ \Lambda_{m_5} : & \quad \left(\bigwedge_{1 \leq j \leq k} (\mathbf{t}_j(i) = \mathbf{t}_j(0)) \right) \end{aligned}$$

The mapping Tr_i , for $i \geq 0$, is defined by induction on the structure of TLTL-formulas.

$$\begin{aligned} \text{Tr}_i(\mathbf{false}) &= \mathbf{false} \\ \text{Tr}_i(p) &= p(i) \\ \text{Tr}_i(x \smile y) &= f(i) \smile g(i) \\ \text{Tr}_i(x \smile t_j + c) &= f(i) \smile \mathbf{t}_j(0) + c \\ \text{Tr}_i(\phi \vee \varphi) &= \text{Tr}_i(\phi) \vee \text{Tr}_i(\varphi) \\ \text{Tr}_i(\bigcirc \phi) &= \text{Tr}_{i+1}(\phi) \\ \text{Tr}_i(\phi \mathcal{U} \varphi) &= \exists j \geq i. (\text{Tr}_j(\varphi) \wedge \forall i \leq k < j. \text{Tr}_k(\phi)) \end{aligned}$$

Given a model $\sigma = s_0, s_1, \dots$ of TLTL-formula ψ , we can associate an \mathcal{L}_2^{TO} interpretation $\mathcal{I} = (\sigma', f, g)$ with $\text{Tr}(\psi)$ by making $p(i) = 1$ if $s_i \models p$ and $f(i) = s_i(x), g(i) = s_i(y)$, and $\mathbf{t}_j(i) = s_0(t_j)$. Similarly, given an \mathcal{L}_2^{TO} interpretation $\mathcal{I} = (\sigma', f, g)$ we generate a model $\sigma = s_0, s_1, \dots$. Now by structural induction on ϕ we can prove the following:

Theorem 1 *Let ψ be a TLTL formula. Then for a given model σ of ψ , we have, $\sigma \models \psi$ if and only if $(\sigma', f, g) \models \text{Tr}(\psi)$.*

4 A Comparison of TLTL with Other Logical Formalisms

The most popular formalism for specifying properties of reactive systems is the linear temporal logic [Pnu77, LP85]. The automatic verification and synthesis for finite state systems is usually carried out using the tableau-based satisfiability algorithm for a propositional version of the linear temporal logic (PLTL) [LP85]. PLTL is interpreted over models which retain only temporal ordering of the states by abstracting away the actual time instants at which events occur. However real-time systems call for explicitly expressing real-time constraints to reason about them, such as the *bounded response* property which necessitates the development of formalisms which can express explicit time.

There are several approaches to extend LTL to express timing constraints. The first approach incorporates an explicit variable x , which expresses the current time without introducing any extra temporal operators. This is referred to as *explicit clock approach*, since the only new element introduced is the explicit clock variable. An example of a first-order explicit clock logic is Real Time Temporal Logic (RTTL) [Ost89], which is defined without restrictions on the assertion language for atomic timing constraints. A propositional version of this logic, called XCTL (Explicit Clock Temporal Logic), is discussed in [HLP90]. This logic allows integer variables to record the values of the global clock at different states, and integer expressions over these variables.

An alternative approach to express timing properties in a temporal logic has been to introduce a bounded version of the temporal operators. For example, a bounded operator $\Diamond_{[2,4]}$ is interpreted as “eventually within 2 to 4 time units”. Using this notation we can write the time-bounded response property discussed earlier as:

$$\Box(p \Rightarrow \Diamond_{[0,5]}q) \quad (\phi_{BR_{MTL}})$$

This approach for the specification of timing properties has been advocated by Koymans [Koy90] and is known as as Metric Temporal Logic (MTL).

In yet another approach, time in a state is accessed through a quantifier, which binds (“freezes”) a variable to the corresponding time. This idea of freeze quantification was introduced by Alur and Henzinger in [AH94] in a logic known as TPTL (Timed Propositional Temporal Logic). The freeze quantifier “ x .” binds the associated variable x to the time of the current temporal context; the formula $x.\phi(x)$ holds at time t_0 iff $\phi(t_0)$ does. Thus in a formula $\Diamond x.\phi$, time variable x is bound to the time of the state at which ϕ is “eventually” true. By admitting atomic formulas that relate the time instants of different states, the time-bounded response property can be written as:

$$\Box x_p.(p \Rightarrow \Diamond x_q.(q \wedge x_q \leq x_p + 5)) \quad (\phi_{BR_{TPTL}})$$

4.1 TLTL vs XCTL

The logic XCTL as described in [HLP90], contains static timing variables and an explicit clock variable in its vocabulary. An atomic formula a_f is either an atomic proposition in \mathcal{P} or a constraint of the form $x \sim u$ or $c \sim u$, where $u = a_0 + a_1 * t_1 + \dots a_m * t_m$ with constants $a_0, a_1 \dots \in \mathbb{N}$ and $c \in \mathbb{N}$, and t_0, t_1, \dots, t_m being static timing variables.

XCTL formulas are built using the following grammar

$$\phi ::= a_f \mid \phi \vee \phi \mid \neg \phi \mid \bigcirc \phi \mid \phi \mathcal{U} \phi$$

where a_f ranges over A_f .

A model for XCTL consists of a sequence of states,

$$\sigma : s_0 s_1 \dots,$$

such that each state s_i gives a boolean interpretation to the propositions and an integer interpretation to the timing variables and to the clock variable x . Similar to TLTL, all static timing variables appearing in a XCTL formula assume the same valuation in all the states.

When compared to TLTL, it turns out that there exist properties involving the dynamic variable y , which cannot be expressed in XCTL. For example, for a timeout model of a real-time system the following property can be expressed in TLTL, - “timeout occurs infinitely often”:

$$\Box \Diamond (x = y) \quad (3)$$

The following sequence of states satisfies (3),

$$\{0, 0\}, \{0, 3\}, \{3, 3\}, \{3, 5\}, \{5, 5\}, \dots \quad (4)$$

In case of XCTL, only way to effectively characterize the state sequences satisfying (3) is by using constraints of the form $x \sim u$ or $u \sim c$. However, since static timing variables need to be given the same value in all the states in a state sequence, an equality of the form $x = u$ involving only static timing variables and constants in the r.h.s. expression u can hold true only for a single value of x (and u) in only finitely many states in a state sequence, where x assumes this value. Therefore, we need an infinite disjunction of such equalities to express (3) in XCTL, implying that there cannot exist any syntactically correct XCTL formula which can effectively characterize the state sequences similar to the one given in (4) satisfying (3).

On the other hand, consider XCTL formula

$$\Box(p \wedge (x = t_p) \Rightarrow \Box(q \wedge (x = t_q) \Rightarrow \Box(r \wedge (x = t_r) \Rightarrow [t_q - t_p \leq t_r - t_q]))) \quad (5)$$

This formula specifies that delay between events p and q is always less than the delay between q and r . This property cannot be specified in TLTL owing to the exclusion of the inequalities involving more than one static timing variable.

4.2 TLTL vs TPTL

In [AH94], Alur and Henzinger proposed an extension of LTL that is capable of relating the times of different states. For this purpose, they use *freeze quantification* by which every variable is bound to the time of a particular state. TPTL allows infinite number of variables $V = \{x_1, x_2, x_3, \dots\}$ over which freeze quantification can be applied. The formulas of TPTL are built using the following grammar,

$$\phi ::= a_f \mid \phi \vee \phi \mid \neg \phi \mid \bigcirc \phi \mid \phi \mathcal{U} \phi \mid x_i.\phi$$

where a_f is either an atomic proposition from \mathcal{P} or a constraint of the form $u_1 \leq u_2$, $u_1 \equiv_d u_2$, where $u_1, u_2 := x_i + c \mid c$ and $c \geq 0, d \geq 2$ are integer constants. Together they form the set of atomic formulas A_f . A variable x_i can be bound by a freeze quantifier as “ $x_i.$ ”, which “freezes” x_i to the time of local temporal context. Only closed formulas, where every occurrence of a variable is under the scope of a freeze quantifier, are considered.

The semantics for TPTL formulas is given by a sequence of states $\sigma = s_0, s_1, \dots$ and an interpretation (environment) for the variables in V , $\mathcal{E} : V \rightarrow \mathbb{N}$. The underlying time domain is taken to be the set of natural numbers \mathbb{N} . As before, each state assigns a Boolean interpretation to the propositions, and a (weakly) monotonic integer interpretation to a (hidden) global timing variable τ , which is not used in the syntax of the formulas. We consider only atomic formulas in A_f and formulas with freeze quantifiers. Let $\mathcal{E}(x_i + c) = \mathcal{E}(x_i) + c$ and $\mathcal{E}(c) = c$. Also let $\mathcal{E}[x_i := a]$ denote the environment that agrees with the environment \mathcal{E} on all variables except x_i , and maps x_i to $a \in \mathbb{N}$.

$$\begin{array}{lll} s_i \models_{\mathcal{E}} p & \text{iff} & s_i(p) = \mathbf{true} \\ s_i \models_{\mathcal{E}} u_1 \leq u_2 & \text{iff} & \mathcal{E}(u_1) \leq \mathcal{E}(u_2) \\ s_i \models_{\mathcal{E}} u_1 \equiv_d u_2 & \text{iff} & \mathcal{E}(u_1) \equiv_d \mathcal{E}(u_2) \\ s_i \models_{\mathcal{E}} x_i.\phi & \text{iff} & s_i \models \phi[\mathcal{E}(x_i) = s_i(\tau)] \end{array}$$

A timed state sequence σ is a model of a closed formula ϕ iff $s_0 \models_{\mathcal{E}} \phi$ for any environment \mathcal{E} .

As already noticed in [AH92], the static constraints in TLTL can play the same role as the freeze quantifier plays in TPTL. For example, consider the time-bounded response property $\phi_{BR_{TLTL}}$ again. This will be satisfied by only those models, which exactly assign a value to t_0 , which is also the clock valuation at the instance of the occurrence of the event p , and therefore it is equivalent to the TPTL formula $\phi_{BR_{TPTL}}$. In general, assuming the same set of atomic constraints, a TPTL formula

$$x.\phi$$

is equivalent to the TLTL formula

$$\forall t_0. (x = t_0 \Rightarrow \phi) \quad (6)$$

However, this apparent syntactic correspondence is not without its problems. TPTL allows defining timing constraints referring to time instances of two past states e.g.,

$$\Box t_1. \bigcirc t_2. \Diamond (alarm \wedge t_2 > t_1 + 5)$$

This formula states that from now, if the time difference between two successive states is more than 5 units, eventually an *alarm* would be raised. Since TLTL does not allow referring to two past time instances, there is no syntactically straightforward translation for such formulas in TLTL using (6) above. However as it turns out, this is really not a problem because such formulas involving reference to two past timing instances are semantically equivalent to formulas which demand referring to only one previous time instance in the state, where the second timing instance would be frozen. In this example an semantically equivalent TPTL formula would be

$$\Box t_1. \bigcirc t_2. (t_2 > t_1 + 5 \Rightarrow \Diamond (alarm))$$

which can be translated into an equivalent TLTL formula using (6) (omitting the outermost quantification)

$$\Box(x = t_1 \Rightarrow \bigcirc(x = t_2 \Rightarrow ((x > t_1 + 5) \Rightarrow \Diamond(alarm))))$$

It may be also noted that TLTL is suitable in case where one needs to express formulas about timed systems with timeouts as the following kinds of condition cannot be expressed in TPTL - “timeout always occurs in the next state of time increment.”

$$\Box((x < y) \Rightarrow \bigcirc(x = y)) \quad (7)$$

The reason that there cannot be any formula in TPTL, which can characterize exactly the same set of models as the formula (7) does is as follows. Since x refers to the time(s) when (7) holds, these can only be captured using freeze quantifier in TPTL. Now since TPTL inequalities only involve (frozen) variables or constants, for variable y also, we need to use these. However, y being a dynamic variable would assume infinitely many different values in a model of the formula (7), these values cannot be captured using constants (or else would demand infinitely many constant based inequalities of the form $[x < c \Rightarrow \bigcirc(x = c)]$). Therefore, the only option is to potentially use variables under freeze quantifier. However, the inequality $x < y$ would demand that such variable (that is y) must refer to a future state, since time flows only in the forward direction, in particular, the next state itself. A formula like the one below may (appear to) capture such a scenario.

$$\Box x.((x < y) \Rightarrow \bigcirc y.(x = y)) \quad (8)$$

However atomic constraints in TPTL cannot refer to the time points of the future states as is evident from the very syntax of the freeze quantifier, e.g., in case of the TPTL formula (8). First, y is a free variable and then y is bound by the (second) freeze quantifier and therefore, both y s are actually different variables - such formulas involving free variables are in any case not allowed in TPTL. Thus, neither constants nor timing variables based inequalities can be used to express the inequalities appearing in the formula (7). That is why, the state sequences satisfying TLTL formula (7) cannot be characterized in TPTL.

On the other hand, there are formulas in TPTL, which cannot be characterized in TLTL. For example, consider the state sequences, in which “an event p occurs at all even time points.” This can be characterized by the TPTL formula $\Box x.(x \equiv_2 0 \Rightarrow p)$. However, as proved in [AH93], this property is not expressible without congruences. This in turn, implies that due to the nature of arithmetical constraints, this TPTL formula cannot be expressed in TLTL.

4.3 TLTL vs MTL

MTL [Koy90] extends LTL by constraining the temporal operators on (bounded or unbounded) intervals of the real numbers specified as subscripts. The formulas in MTL are inductively built using the following grammar

$$\phi ::= p \mid \phi \vee \phi \mid \neg \phi \mid \phi \mathcal{U}_I \phi$$

where $p \in \mathcal{P}$ is a proposition and I is a (bounded or unbounded) interval with integer (or rational) end-points. An *interval* is a nonempty convex subset of

$\mathbb{R}^{\geq 0}$, which may assume one of the following forms: $[a, b]$, $[a, b)$, $[a, \infty)$, $(a, b]$, (a, b) , (a, ∞) , where $a \leq b$ for $a, b \in \mathbb{R}^{\geq 0}$. The interval I is singular *iff* it is of the form $[a, a]$ (also written as $= a$).

The formulas of MTL can be interpreted over a timed state sequence (σ, f) , where $\sigma = s'_0, s'_1, \dots$ is a untimed state sequence giving Boolean interpretation to the propositions and $f : \mathbb{N} \mapsto \mathbb{R}^{\geq 0}$ is a mapping such that $f(i)$ denotes the time at state s'_i . The satisfaction relation $(\sigma, f) \models \phi$ is defined in a usual way. We only mention the case of the formula $\phi \mathcal{U}_I \varphi$:

$$(s_i, f(i)) \models \phi \mathcal{U}_I \varphi \text{ iff } \exists j \geq i. ((s_j, f(j)) \models \varphi) \\ \bigwedge (\forall i \leq k < j. (s_k, f(k)) \models \phi \wedge (f(j) \in f(i) + I)),$$

where $f(i) + I$ is defined using simple rules of interval arithmetic, e.g., if $I = [a, b]$, then $f(i) + I$ stands for the interval $[f(i) + a, f(i) + b]$.

Since it is well known that the satisfiability and model-checking problems for MTL are undecidable over the state-based semantics (under $\mathbb{R}^{\geq 0}$), we will consider a fragment of MTL known as *Metric Interval Temporal Logic* (MITL) introduced by Alur et al. [AFH96], in which the temporal operators can only be constrained by nonsingular intervals. Thus ‘punctuality properties’ like $\Diamond_{=3} p$ (“eventually exactly after 3 time units p would hold”) cannot be specified in MITL.

It is known that any MITL formula can also be expressed in TPTL [AH93]. Specifically, if the atomic constraints permit comparison and addition of constants, then MITL formula

$$\phi \mathcal{U}_I \varphi \tag{9}$$

is equivalent to the TPTL formula

$$x. \phi \mathcal{U} z. (\varphi \wedge z \in x + I)$$

where $z \in x + I$ can be expressed using TPTL constraints given the boundaries of I . It has been shown recently in [BCM05] that TPTL is strictly more expressive than MTL for both point-wise and interval-based semantics. Now in the light of the discussion presented in previous Section 4.2, it is easy to see that any MITL formula can also be expressed in TLTL. Specifically, MITL formula (9) is equivalent to the TLTL formula

$$\forall t_0. (x = t_0 \Rightarrow \phi \mathcal{U} (\varphi \wedge x \in t_0 + I))$$

where $x \in t_0 + I$ can be expressed using atomic constraints in TLTL, given the boundaries of I . For example, MITL formula, $\Box(p \Rightarrow \Diamond_{[2,5]} q)$ can be expressed in TLTL as

$$\forall t_0. \Box(p \wedge x = t_0 \Rightarrow \Diamond(q \wedge x \geq t_0 + 2 \wedge x \leq t_0 + 5)).$$

Also, on the other hand, there exist TLTL formulas (e.g., one given in (7)), which cannot be expressed in MTL under point-wise semantics.

5 Decision Procedure for Validity of TLTL formulas

We consider a decision procedure for checking the validity of TLTL formulas employing similar techniques used in [HLP90]. In order to check the validity

of a given TLTL formula $\psi = \forall t_1 \dots t_k. \phi$, we take the negated formula $\neg\phi$ and actually check for its satisfiability using a tableau like construction by posing the question, ‘are there positive real values for the timing variables t_1, \dots, t_k that will make the formula $\neg\phi$ satisfiable?’

5.1 Closure of a Formula

Let ϕ be a TLTL formula, which is to be checked for satisfiability. We define the Fischer-Ladner closure $Cl(\phi)$ as the least set containing ϕ and closed under the following:

- (c_1) $\text{true}, \text{false}, \bigcirc\text{true} \in Cl(\phi)$,
- (c_2) $\forall p \in \mathcal{P}_\phi, p, \neg p \in Cl(\phi)$, where \mathcal{P}_ϕ is the set of atomic propositions appearing in ϕ ,
- (c_3) $\neg\psi \in Cl(\phi) \Leftrightarrow \psi \in Cl(\phi)$ – we identify $\neg\neg\psi$ with ψ and $\neg\text{true}$ with false ,
- (c_4) $\psi \vee \psi' \in Cl(\phi) \Rightarrow \psi, \psi' \in Cl(\phi)$,
- (c_5) $\bigcirc\psi \in Cl(\phi) \Rightarrow \psi \in Cl(\phi)$,
- (c_6) $\neg \bigcirc \psi \in Cl(\phi) \Rightarrow \bigcirc\neg\psi \in Cl(\phi)$,
- (c_7) $\psi \mathcal{U}\psi' \in Cl(\phi) \Rightarrow \psi, \psi', \bigcirc(\psi \mathcal{U}\psi') \in Cl(\phi)$,
- (c_8) $x \smile y \in Cl(\phi) \Rightarrow x < y, x = y \in Cl(\phi)$,
- (c_9) $x \smile u \in Cl(\phi) \Rightarrow x \smile' u \in Cl(\phi)$ for every $\smile' \in \Delta$,
- (c_{10}) $x < y \in Cl(\phi) \Rightarrow \bigcirc(x = y), \diamond(x < y) \in Cl(\phi)$
- (c_{11}) $x = y \in Cl(\phi) \Rightarrow \bigcirc(x < y), \diamond(x = y) \in Cl(\phi)$
- (c_{12}) $x \smile u \in Cl(\phi) \Rightarrow \diamond(x > u) \in Cl(\phi)$.

Intuitively $Cl(\phi)$ includes all the formulae that play some role in deciding the satisfiability of ϕ . Using structural induction on ϕ , it can be shown that $|Cl(\phi)| \leq 7|\phi| + 3$.

5.2 Atoms

An atom $A \subseteq Cl(\phi)$ is a consistent set of formulas such that

- (a_1) $\text{true}, \bigcirc\text{true} \in A$.
- (a_2) For every $\psi \in A \Leftrightarrow \neg\psi \notin A$.
- (a_3) For every $\psi \vee \psi' \in A \Leftrightarrow \psi \in A$ or $\psi' \in A$.
- (a_4) For every $\psi \mathcal{U}\psi' \in A \Leftrightarrow \psi' \in A$ or $\psi, \bigcirc(\psi \mathcal{U}\psi') \in A$.
- (a_5) For every $x < y, x = y \in Cl(\phi)$, precisely one of them is in A .
- ($a_{6.1}$) For every $x < y \in A \Rightarrow \bigcirc(x = y) \in A$.
- ($a_{6.2}$) For every $x = y \in A \Rightarrow \bigcirc(x < y) \in A$.

- (a₇) For every $x \smile u \in Cl(\phi)$, exactly one of $x < u$, $x = u$, or $x > u$ is in A .
- (a₈) If $C(A)$ denotes the set of all constraints in A , it is required that $C(A)$ forms a consistent set. In particular, for every $x \smile u_i \in Cl(\phi)$, $x \smile u_i \in C(A)$ only if exactly one of the following holds, where we let $H_A = \bigwedge_{x \smile u_j \in C(A)} (x \smile u_j)$:
- $$\begin{cases} x < y \in A \text{ and } (x < y) \wedge (x \smile u_i) \wedge H_A \text{ is satisfiable over } \mathbb{R}^{\geq 0} & \text{OR} \\ x = y \in A \text{ and } (x = y) \wedge (x \smile u_i) \wedge H_A \text{ is satisfiable over } \mathbb{R}^{\geq 0} & \text{OR} \\ x < y, x = y \notin A \text{ and } (x \smile u_i) \wedge H_A \text{ is satisfiable over } \mathbb{R}^{\geq 0} \end{cases}$$
- Informally, we include a static constraint $x \smile u_i \in Cl(\phi)$ in atom A only if the resultant set of constraints in A remains consistent.
- (a₉) For every $x \smile u \in A \Rightarrow \text{true } \mathcal{U}(x > u) \in A$.

The requirement that every atom contains the formula $\bigcirc \text{true}$ is to ensure that only infinite sequences will be considered as possible models.

Additionally, we define two special atoms.

$$\begin{aligned} A_{0=} &= \{\text{true}, \bigcirc \text{true}, x = 0, x = y, \bigcirc(x < y), \text{true} \mathcal{U}(x > 0), \\ &\quad \bigcirc(\text{true} \mathcal{U}(x > 0))\}, \text{ and} \\ A_{0<} &= \{\text{true}, \bigcirc \text{true}, x = 0, x < y, \bigcirc(x = y), \text{true} \mathcal{U}(x > 0), \\ &\quad \bigcirc(\text{true} \mathcal{U}(x > 0)), \bigcirc(x > 0)\}. \end{aligned}$$

We denote the set of all atoms by At , which also contains $A_{0=}$ and $A_{0<}$.

5.3 Tableau Construction

We construct a structure $\mathcal{A}_\phi = (At, R)$, which is a directed graph with atoms as nodes; and its edges are defined by the relation R as follows:

$$(A, B) \in R \Leftrightarrow \begin{cases} 1. & \text{for every } \bigcirc a \in Cl(\phi), \\ & \bigcirc a \in A \Leftrightarrow a \in B, \text{ where } a \in \mathcal{P}_\phi \cup C(\phi); \\ 2. & \text{for every } x = u \in Cl(\phi), \\ & x = u \in A \Rightarrow x = u \in B \text{ or } x > u \in B; \\ 3. & \text{for every } x > u \in Cl(\phi), \\ & x > u \in A \Rightarrow x > u \in B; \end{cases}$$

where $C(\phi)$ refers to the set of atomic constraints appearing in ϕ .

It is not difficult to see that under the definition of R , the following facts hold.

Fact 2 *There is no atom $A \in At$ such that $(A, A_{0=}) \in R$.*

Fact 3 *There is no atom $A \in At \setminus \{A_{0=}\}$ such that $(A, A_{0<}) \in R$.*

In other words, atom $A_{0=}$ has no incoming edges and the only permissible incoming edge to atom $A_{0<}$ is $(A_{0=}, A_{0<}) \in R$. $A_{0=}$ and $A_{0<}$ will be referred from now on as *initial atoms*. Also note that only states, where atom $A_{0=}$ may hold are those which interpret both clock variable x and minimum of the timeout variable y as 0.

Let $\mathcal{A}' = (W', R')$ be a substructure of \mathcal{A}_ϕ and let \mathcal{C} be a strongly connected subgraph (SCS) of \mathcal{A}' .

- \mathcal{C} is said to be *terminal* in \mathcal{A}' if it has no outgoing edges.
- \mathcal{C} is said to be *self-fulfilling* if every atom has a successor in \mathcal{C} , and for every $p \mathcal{U} q \in A \in \mathcal{C}$, there exists $B \in \mathcal{C}$ such that $q \in B$.
- \mathcal{C} is said to be *useless* in \mathcal{A}' if it is terminal in \mathcal{A}' but is not self-fulfilling.

5.4 The Timing Relation between Atoms

Relation between Successive Atoms:

Consider two atoms A, B from \mathcal{A}_ϕ such that $(A, B) \in R$. Assume the set of constraints in A to be $C(A) = T(A) \cup S(A)$, where $T(A) = \{T_{out}\}$ contains the (unique) dynamic constraint and $S(A) = \{S_1, \dots, S_m\}$ the set of static constraints. Further, the set of constraints in B is $C(B) = T(B) \cup S(B)$ where $T(B) = \{T'_{out}\}$, and $S(B) = \{S'_1, \dots, S'_m\}$. For every S_i there is a corresponding S'_i and for T_{out} there is a corresponding T'_{out} such that:

- if S_i is $x < u$, S'_i is $x \sim u$. This follows from the condition (a_7) in Section 5.2 for defining an atom,
- if S_i is $x = u$, S'_i is either $x = u$ or $x > u$. This follows from the condition (2) for defining R in Section 5.3,
- if S_i is $x > u$, S'_i is also $x > u$. This follows from the condition (3) for defining R in Section 5.3, and
- if T_{out} is $x < y$, T'_{out} is $x = y$. Else if, T_{out} is $x = y$, T'_{out} is $x < y$. This follows from conditions $(a_{6.1}), (a_{6.2})$ in Section 5.2 and condition (1) for defining R in Section 5.3.

The temporal relation between two atoms produces the following results, which allow us to select values for x, y satisfying constraints in one atom, once the values for which these variables satisfy other constraints are known. Let us assume that χ, χ' denote valuations for clock x , ψ, ψ' for y , and $\alpha_1, \alpha_2, \dots, \alpha_k$ for timing variables t_1, t_2, \dots, t_k .

Lemma 2 *If $\chi', \psi', \alpha_1, \alpha_2, \dots, \alpha_k$ are non negative reals satisfying $C(B)$, there exist non negative reals χ, ψ such that $\chi, \psi, \alpha_1, \alpha_2, \dots, \alpha_k$ satisfy $C(A)$ and $\chi \leq \chi', \psi \leq \psi'$.*

Proof. Assume $\chi', \psi', \bar{\alpha}$ satisfy $C(B)$, where $\bar{\alpha} = \alpha_1, \alpha_2, \dots, \alpha_k$. We need to show that there exist $\chi \leq \chi', \psi \leq \psi'$ such that $\chi, \psi, \bar{\alpha}$ satisfy $C(A)$. We consider different cases.

Case 0: If $C(A) = \emptyset$, that is, ϕ is a purely qualitative formula not involving any of static or dynamic constraints, choose $\chi = \chi'$ and $\psi = \psi'$.

Case 1: If $S(A) = \emptyset$ but $T(A) \neq \emptyset$. We choose χ, ψ based upon the nature of T_{out} .

- Let $T_{out} \equiv x = y \in C(A)$. Now by the definition of timing relation between atoms A and B , we have $T'_{out} \equiv x < y$ implying that $\chi' < \psi'$. So, choose $\psi = \chi = \chi'$.

- Let $T_{out} \equiv x < y \in C(A)$. Again by the definition of timing relation between atoms A and B , we have $T'_{out} \equiv x = y \in C(B)$. Therefore $\chi' = \psi'$. We choose $\psi = \psi'$ and some arbitrary value $\chi \in [0, \chi')$. Note that this is always feasible since in the only exceptional case when $\chi' = \psi' = 0$, B would be an initial atom $A_{0=}$ and thus A cannot be present (see Fact 2).

Case 2: $S(A) \neq \emptyset$ and there exists a constraint $S_i \in S(A)$ of the form $x = t_i + c_i$ or $x = c$ (c_i, c are constants) as the case may be, then choose χ as $\alpha_i + c_i$ or c which would necessarily satisfy all of S_1, \dots, S_m following the definition of an atom (condition (a_8)). Now based upon the nature of T_{out} , we will choose ψ and prove the consistency of the choice.

- Let $T_{out} \equiv x = y \in C(A)$. Choose $\psi = \chi$. Now by the definition of timing relation between atoms A and B , we have $T'_{out} \equiv x < y$. Therefore $\chi' < \psi'$. Now $(x = t_i + c_i) \in S(A) \Rightarrow (x = t_i + c_i) \text{ or } (x > t_i + c_i) \in S(B)$ implying $\chi' \geq \alpha_i + c_i$. Thus we have, $\psi = \chi = \alpha_i + c_i \leq \chi' < \psi'$. Similarly, for $(x = c) \in S(A)$.
- Let $T_{out} \equiv x < y \in C(A)$. Choose ψ such that $\alpha_i < \psi \leq \psi'$. Again by the definition of timing relation between atoms A and B , we have $T'_{out} \equiv x = y \in C(B)$. Therefore $\chi' = \psi'$. Also $(x = t_i + c_i) \in C(A) \Rightarrow (x = t_i + c_i) \text{ or } (x > t_i + c_i) \in C(B)$, which also means $\chi' \geq \alpha_i + c_i$, i.e., $\chi' \geq \chi$. Similarly, for $(x = c) \in S(A)$.
- $T(A) = \emptyset$. Choose $\psi = \chi$.

So, in all the situations we can choose χ and ψ such that $\chi \leq \chi'$ and $\psi \leq \psi'$.

Case 3: $S(A) \neq \emptyset$ and there does not exist any constraint $S_i \in S(A)$ of the form $x = t_i + c_i$ or $x = c$. Let

- $E_l = \{\alpha_j + c_j \mid (x > t_j + c_j) \in C(A)\} \cup \{c \mid (x > c) \in C(A)\}$ and $l = \max(E_l)$ if $E_l \neq \emptyset$ else $l = -\infty$,
- $E_m = \{\alpha_j + c_j \mid (x < t_j + c_j) \in C(A)\} \cup \{c \mid (x < c) \in C(A)\}$ and $m = \min(E_m)$ if $E_m \neq \emptyset$ else $m = \infty$.

Note that $l < m$ since $\bigwedge_i S_i$ is satisfiable. Again, by the definition of timing relation between atoms A and B , we have $\forall w \in E_l. (x > w) \in S(A) \Rightarrow (x > w) \in S(B)$ implying that $l < \chi'$. Therefore, choose χ such that

$$\begin{aligned} l < \chi \leq \chi' & \text{ if } \chi' < m \\ l < \chi < m & \text{ if } \chi' \geq m \end{aligned} \tag{10}$$

Such a choice of χ satisfies all of S_1, \dots, S_m . Now based upon the nature of T_{out} , we choose the values of χ and ψ and prove the consistency of such a choice.

- Let $T_{out} \equiv x = y \in C(A)$. Choose any value for χ satisfying (10) and choose $\psi = \chi$. Since $\chi \leq \chi' < \psi'$, we have $\chi \leq \chi'$ and $\psi < \psi'$.
- Let $T_{out} \equiv x < y \in C(A)$. Choose $\psi = \chi'$. Because $T_{out} \wedge \bigwedge_i S_i$ is satisfiable, we must be able to choose χ such that $\chi < \psi$, which implies that $\chi < \chi'$.
- $T(A) = \emptyset$. Choose any value for χ satisfying (10) then choose $\psi = \chi$.

So in both the situations we can choose χ and ψ such that $\chi \leq \chi'$ and $\psi \leq \psi'$. Hence. ■

Relation between Atoms in a Self-Fulfilling SCS:

In a self-fulfilling SCS every two atoms have the same set of static constraints, but they differ in the dynamic constraint.

Lemma 3 *Let A and B be two atoms in some self-fulfilling SCS \mathcal{C} , then $S(A) = S(B)$, and all the static constraints must be of the form $x > u$.*

Proof. Since $A, B \in \mathcal{C}$ and \mathcal{C} is a SCS, hence by definitions of atom and relation R , $x > u \in S(A) \Leftrightarrow x > u \in S(B)$. It remains to show that $(x \smile' u) \notin S(A)$, where $\smile' \in \{<, =\}$. Assume that it is not the case, which means, $x \smile' u \in C(A)$. By the definition of an atom, $\text{true } \mathcal{U}(x > u) \in A$. Since \mathcal{C} is a self-fulfilling SCS, there must be an atom $D \in \mathcal{C}$ such that $(x > u) \in S(D)$. It follows that $(x > u) \in S(A)$ as well because A is reachable from D , a fact that contradicts the definition of an atom. Therefore, we conclude that $x \smile' u \notin S(A)$. Since this will be true for atom B as well, it follows $S(A) = S(B)$. ■

Lemma 4 *If $\chi, \psi, \bar{\alpha}$ is a satisfying solution for $C(A)$ and $A \in \mathcal{C}$ (a self-fulfilling SCS), for every $B \in \mathcal{C}$ such that $(A, B) \in R$, there exist χ', ψ' , such that $\chi', \psi', \bar{\alpha}$ satisfy $C(B)$ and $\chi' \geq \chi, \psi' \geq \psi$.*

Proof. We consider only dynamic constraints appearing in A and B :

- $(x = y) \in C(A) \wedge (x < y) \in C(B)$. Choose $\chi' = \chi$ and any $\psi' > \psi$.
- $(x < y) \in C(A) \wedge (x = y) \in C(B)$. Choose $\chi' = \psi' = \psi$.
- $T(A) = T(B) = \emptyset$. Choose arbitrarily $\chi', \psi' \in \mathbb{R}^{\geq 0}$ such that $\chi' > \chi, \psi' > \psi$, and $\chi' \leq \psi'$.

Note that from Lemma 2, every atom in \mathcal{C} contains all other constraints of the same form $x > u$, which are immediately satisfiable by any $\chi' \geq \chi$. ■

5.5 Fulfilling Paths and Satisfiability

An infinite path $\pi = A_0, A_1, \dots$, (where A_0, A_1, \dots are atoms) is called a *fulfilling path* for ϕ if for every $i \geq 0$:

1. $\phi \in A_0$.
2. $(A_i, A_{i+1}) \in R$.
3. For every $p \mathcal{U} q \in Cl(\phi)$, if $p \mathcal{U} q \in A_i$, then there exists some $j \geq i$ such that $q \in A_j$.

Theorem 5 *The formula ϕ is satisfiable if and only if there exists a fulfilling path for ϕ in \mathcal{A}_ϕ .*

Proof. If ϕ is satisfiable and σ is a model for it then the corresponding fulfilling path can be given by $\pi = A_0, A_1, \dots$, where $A_i = \{p \in Cl(\phi) \mid \sigma^i \models p\}$.

On the other hand let $\pi = A_0, A_1, \dots$, be a fulfilling path for ϕ . Define a model $\sigma = s_0, s_1, \dots$, for ϕ such that each state s_i ($\forall i \geq 0$), interprets proposition p as **true** iff p is in A_i . Since π is an infinite path, beyond a certain point

(say A_k), all the atoms in π must be repeating infinitely often. These infinitely repeating atoms must be reachable from each other, and hence must be contained in a self-fulfilling SCS \mathcal{C} . Let $\alpha_1, \alpha_2, \dots, \alpha_k, s_{k+1}(x), s_{k+1}(y)$ be any solution that satisfies $C(A_{k+1})$. Using Lemma 2, we can trace the path π backwards till A_0 assigning values ($s_0(x) \leq s_1(x) \leq \dots \leq s_{k-1}(x) \leq s_k(x) \leq s_{k+1}(x), s_0(y) \leq s_1(y) \leq \dots \leq s_{k-1}(y) \leq s_k(y) \leq s_{k+1}(y)$) to (x, y) in atoms A_0, A_1, \dots, A_k on the way, which satisfy constraints in $C(A_0), C(A_1), \dots, C(A_k)$. Also using Lemmas 3,4 we can assign values ($s_{k+1}(x) \leq s_{k+2}(x) \leq s_{k+3}(x) \leq \dots, s_{k+1}(y) \leq s_{k+2}(y) \leq s_{k+3}(y) \leq \dots$) for the future states s_{k+2}, s_{k+3}, \dots . Clearly σ is a infinite sequence of states satisfying the formula ϕ . ■

From this theorem we conclude that it is sufficient to look for a fulfilling path for ϕ in \mathcal{A}_ϕ in order to determine the satisfiability of ϕ .

5.6 Satisfiability Checking

The fulfilling path for a TLTL formula ϕ can be constructed as follows:

let $\mathcal{A}^* = (\mathcal{W}^*, \mathcal{R}^*) = \mathcal{A}_\phi$ be the initial structure resulting from the construction described in the Section 5.3.

while ($\mathcal{A}^* \neq \emptyset$ OR \mathcal{A}^* does not contain any useless maximal SCS)

begin

let \mathcal{C} be a useless maximal SCS in \mathcal{A}^*

$\mathcal{W}^* = \mathcal{W}^* \setminus \mathcal{C}$

$\mathcal{R}^* = \mathcal{R}^* \cap (\mathcal{W}^* \times \mathcal{W}^*)$

end

if (there is an atom A in \mathcal{W}^* such that $\phi \in A$)

then report **success**

else report **failure**.

Theorem 6 *The formula ϕ is satisfiable if and only if the above algorithm reports success.*

The algorithm succeeds if and only if the tableau \mathcal{A}_ϕ contains a finite path $\pi = A_0, \dots, A_k$ that starts at an atom A_0 , containing ϕ , and reaches A_k at a terminal self-fulfilling SCS \mathcal{C} . This path can be used to construct a fulfilling path for ϕ . Hence by Theorem 5, ϕ is satisfiable if and only if the algorithm above reports success. ■

5.7 Complexity Analysis

For the complexity analysis we would require the following result.

Lemma 7 *Checking that the constraints appearing in an atom are satisfiable over $\mathbb{R}^{\geq 0}$ can be done in time $O(|Cl(\phi)|)$.*

Proof. There exists a well known polynomial time procedure [Pra77] to decide the satisfiability of a conjunction of linear inequalities of the form $\xi \leq \eta + c$, where ξ, η are real-valued variables and c is an integer constant, by reducing the problem to the problem of deciding the nonexistence of a cycle with negative weight in a weighted directed graph such that inequality $\xi \leq \eta + c$ induces two nodes corresponding to variables ξ, η and an edge (ξ, η) labeled with $-c$.

Nonetheless, owing to special nature of the constraints considered here, we can show that a linear time procedure exists to check the satisfiability of the constraints appearing in an atom. Let us partition the the set of constraints appearing in atom A as follows:

$$C(A) = C_{xy} \cup C_{=c} \cup C_{=v} \cup C_{>c} \cup C_{>v} \cup C_{<c} \cup C_{<v}, \text{ where}$$

- C_{xy} consists of constraints of the form $(x \sim y)$,
- $C_{=c}$ consists of constraints of the form $(x = c)$,
- $C_{=v}$ consists of constraints of the form $(x = t + c')$,
- $C_{>c}$ consists of constraints of the form $(x > c)$,
- $C_{>v}$ consists of constraints of the form $(x > t + c')$,
- $C_{<c}$ consists of constraints of the form $(x < c)$, and
- $C_{<v}$ consists of constraints of the form $(x < t + c')$.

Note $\sim \in \{<, =\}$, and $c, c' \in \mathbb{N}$ are integer constants, and $t \in T$ is a timing variable.

If $|C_{=c}| > 1$, then $C_{=c}$ itself is unsatisfiable and so is $C(A)$. Otherwise if $(x = c) \in C_{=c}$ then check whether constraints in $C_{>c} \cup C_{<c}$ are satisfiable on assigning c to x . If not, then $C(A)$ is also not satisfiable. Otherwise, $\forall t_1 \in T$ such that $(x = t_1 + c_1) \in C_{=v}$, we can assign valuation $c - c_1$ for t_1 ; $\forall t_2 \in T$ such that $(x < t_2 + c_2) \in C_{<v}$, we can assign valuation $(c - c_2) + z$, ($z > 0 : (c - c_2) + z > 0$) for t_2 ; and $\forall t_3 \in T$ such that $(x > t_3 + c_3) \in C_{>v}$, we can assign $(c - c_3) - z$, ($z > 0 : (c - c_3) - z > 0$) to t_3 . Also assign c to y if $(x = y) \in C_{xy}$, else assign $c + 1$.

In the other case, when $C_{=c} = \emptyset$, calculate $l = \max(C_{>c})$ if $C_{>c} \neq \emptyset$, else $l = -\infty$ and $m = \min(C_{<c})$ if $C_{<c} \neq \emptyset$, else $m = \infty$. We define $\max(C_{>c}) = \max\{c \in \mathbb{R}^{\geq 0} \mid x > c \in C_{>c}\}$, and $\min(C_{<c}) = \min\{c \in \mathbb{R}^{\geq 0} \mid x < c \in C_{<c}\}$. Next we check if $l < m$. If not, these constraints cannot be satisfied simultaneously. Otherwise we can choose any value of x , $l < x < m$, as a solution. Satisfying valuations to all timing variables can be assigned accordingly.

To estimate the time complexity, notice that partitioning of $C(A)$ can be done in linear time with respect to the size of the constraint set since in order to place a constraint in its correct partition it only requires to check the form of inequality and type of variable (constant or variable). All other steps of checking satisfiability and assigning valuations to timing variables in T can also be carried out in time linear on the size of the constraint set, where size of the constraint set is bounded by $|Cl(\phi)|$ ■

Theorem 8 *Satisfiability problem for (unquantified) TLTL is PSPACE Complete.*

Proof. Let $|\mathcal{A}_\phi|$ denote the size of the structure \mathcal{A}_ϕ , which is bounded by the number of possible subsets of $Cl(\phi)$, that is, $|\mathcal{A}_\phi| \leq 2^{\mathcal{O}(|Cl(\phi)|)}$. The number of constraints appearing in any atom are also bounded by $|Cl(\phi)| \leq 7|\phi|$, therefore $|\mathcal{A}_\phi| \leq 2^{\mathcal{O}(|\phi|)}$. By Lemma 7, consistency checking of these constraints can be performed in time $\mathcal{O}(|Cl(\phi)|)$. This results in an overall time-bound $2^{\mathcal{O}(|\phi|)}|\phi| = 2^{\mathcal{O}(|\phi| + \log|\phi|)} = 2^{\mathcal{O}(|\phi|)}$.

Using a similar argument presented in [SC85], we can conclude that there exists a nondeterministic algorithm M , which (generates \mathcal{A}_ϕ ‘on-the-fly’ and) accepts ϕ iff it is satisfiable.

M uses space of the order of $|Cl(\phi)|$. Using Savitch Theorem [Sav70], it can be concluded that there exists a polynomial space bounded ($\mathcal{O}(|Cl(\phi)|^2)$) deterministic algorithm which can decide satisfiability of a TLTL formulae.

It is also shown in [SC85] that satisfiability of LTL with \mathcal{U} and \bigcirc is PSPACE-hard. Since LTL is properly embedded in TLTL, it renders satisfiability of (unquantified) TLTL PSPACE-complete. ■

As a consequence, we also have,

Theorem 9 *Validity problem for (quantified) TLTL is PSPACE Complete.*

6 Model Checking for TLTL

The model checking problem of deciding whether a TLTL formula ψ is satisfied by all the computations of a given timeout program P with clock, timeout, and static timing variables, is conceptually much harder than deciding the validity of TLTL formulas. This difficulty arises due to the fact that clock, timeout, and static timing variables range over the set of non negative reals, and therefore timeout systems are inherently infinite state systems. This render automated verification of these systems difficult as most of the model checking techniques proceed by exhaustive enumeration of the state space.

Therefore we consider a restriction of TLTL over \mathbb{N} (i.e., clock, timeout, and static timing variables assume positive integer valuations). Also we restrict our attention to only those timeout systems where increments in the values of the timeout variables and thus, the clock increments are allowed only over a finite range of values, while taking transitions.

6.1 Timeout Programs

The representation of a finite state timeout program that we consider, is given by a *timeout Kripke structure* (TKS) $K = \langle S, S^0, E \rangle$ over the clock x , the set of static timing variables T , a finite set \mathcal{TO} of timeout variables $\tau_1, \tau_2, \dots, \tau_n$ used to record the values of timeouts such that $\mathcal{TO} \cap T = \emptyset$, and a variable y which equals $\min \mathcal{TO} = \min\{\tau_i : \tau_i \in \mathcal{TO}\}$, where

- S is a finite set of locations. Each location $s \in S$ gives a boolean interpretation to each of the propositions and an integer interpretation to static timing variables appearing in ψ (i.e., the set T_ψ) in the interval $[0, M]$,
- $S^0 \subseteq S$ is the set of initial locations defining the values for static timing variables for the runs starting from these locations,
- $E = (E^+ \cup E^0) \subseteq (S \times \mathbb{N} \times (\mathbb{N} \cup \{\star\}) \times S)$ - denotes the set of edges connecting locations in S . E is partitioned into two disjoint sets E^+ and E^0 . If $(s, l, m, s') \in E^+$ then $l = m = 0$. For simplicity we omit l and m for E^+ edges and represent them as (s, s') . For E^0 edges either l and m assume non zero positive integral values, which define the finite range of values for incrementing timeouts or, specifies open ended range of values larger than l for incrementing timeouts when m is \star .

The operational meaning to E^+ and E^0 is given as follows.

- E^+ is the set of delay transitions, whereby clock x advances to $\min \mathcal{TO}$, that is, if $(s, 0, 0, s') \in E^+$, then on taking this transition, the value of the clock x is incremented to $\min \mathcal{TO}$.
- E^0 represents the set of the discrete transitions. For $(s, l, m, s') \in E^+$ on a discrete transition at least one of the timeouts attaining the minimum value is incremented by some arbitrary value δ in $[l, m]$ (if $m \in \mathbb{N}$), or $\delta \geq l$ (if m is \star).

The semantics of a TKS K is defined as follows: We define a timeout computation of K to be an infinite sequence of timeout states

$$\sigma : \langle s_0, x_0, y_0, \mathcal{TO}_0 \rangle, \langle s_1, x_1, y_1, \mathcal{TO}_1 \rangle, \dots,$$

where x_0, x_1, \dots denote the clock values, y_0, y_1, \dots denote the values for the variable y , and $\mathcal{TO}_0, \mathcal{TO}_1, \dots$ denote sets of values for the timeouts in \mathcal{TO} for $i = 0, 1, \dots$ such that $\mathcal{TO}_i[j]$ would denote the value of τ_j in \mathcal{TO}_i . All (static) timing variables in T assume the same valuation in every state. Thus we have,

- $s_0 \in S_0$ and either $x_0 = y_0 = \min \mathcal{TO}_0 = 0$ or $0 = x_0 < y_0 = \min \mathcal{TO}_0$.
- For every $i = 0, 1, \dots$
 - $\forall t_j \in T : s_i(t_j) = s_0(t_j)$
 - $y_i = \min \mathcal{TO}_i$.
- For every $i = 0, 1, \dots$
 - either $(s_i, 0, 0, s_{i+1}) \in E^+$, s.t. $x_i < \min \mathcal{TO}_i \wedge x_{i+1} = \min \mathcal{TO}_i$. Also $\mathcal{TO}_{i+1} = \mathcal{TO}_i$, that is, during delay transitions timeouts do not change.
 - or $(s_i, l, m, s_{i+1}) \in E^0$ and $\exists \tau_j \in \mathcal{TO}$ s.t. $\mathcal{TO}_i[j] = \min \mathcal{TO}_i$, and $\mathcal{TO}_{i+1}[j] = \mathcal{TO}_i[j] + \delta$ where $\delta \in [l, m]$ if $m \in \mathbb{N}$, otherwise $\delta \geq l$ if m is \star . Also $x_{i+1} = x_i = \min \mathcal{TO}_i$ and $\forall \tau_k \in \mathcal{TO} \setminus \{\tau_j\} . \mathcal{TO}_{i+1}[k] = \mathcal{TO}_i[k]$.
- There are infinitely many i 's such that $x_{i+1} = \min \mathcal{TO}_i$, which means clock and timeouts always advance.

6.2 A Tableau Construction for the Product of the program K and the formula ϕ

We construct a tableau $\mathcal{K} = \mathcal{A}_{\phi \times K}$ as the cross product of the tableau for a (unquantified) satisfiable TLTL formula ϕ and a TKS K . The elements of \mathcal{K} are

- $N_{\mathcal{K}}$ is the set of the nodes consisting of pairs $\langle A, s \rangle$ with $A \in \mathcal{A}_{\phi}$ (tableau for ϕ) and $s \in K$.
- $E_{\mathcal{K}} = E_{\mathcal{K}}^+ \cup E_{\mathcal{K}}^0$ is the transition relation where $E_{\mathcal{K}}^+$ captures the elapse of time and $E_{\mathcal{K}}^0$ represents the discrete transition. Let $u ::= t + c \mid c$, which is defined in Section 2.1.
 - $(\langle A, s \rangle, \langle A', s' \rangle) \in E_{\mathcal{K}}^+$ iff $(A, A') \in R, (s, 0, 0, s') \in E^+$ and

$$\begin{aligned}
& x < u \in C(A) \Rightarrow x = u \in C(A') \text{ or } x > u \in C(A'), \\
& x = u \in C(A) \Rightarrow x > u \in C(A'), \text{ and} \\
& x < y \in C(A) \Rightarrow x = y \in C(A') \\
& - (\langle A, s \rangle, \langle A', s' \rangle) \in E_K^0 \text{ iff } (A, A') \in R \text{ and } (s, l, m, s') \in E^0 \text{ and} \\
& \quad x \smile u \in C(A) \Leftrightarrow x \smile u \in C(A') \\
& \quad x = y \in C(A) \Rightarrow (x < y) \in C(A')^4
\end{aligned}$$

- N_0 is the set of initial nodes consisting of all pairs $\langle A, s \rangle$ such that $\phi \in A$ and $s \in S_0$.

6.3 Model Checking Procedure

We check if all runs of a program K satisfy a TLTL-formula $\psi = \forall t_1 \dots t_k. \phi$ as follows:

Step1 Construct the initial tableau $\mathcal{A}_{\neg\phi}$ for the negated formula $\neg\phi$ as described in Section 5.

Step2 Construct the tableau product $\mathcal{A}_{\neg\phi \times K}$ as described in the Section 6.2.

Step3 Check if $\mathcal{A}_{\neg\phi \times K}$ contains a self-fulfilling path for $\neg\phi$.

Lemma 10 *The TKS K satisfies $\neg\phi$ if and only if $\mathcal{A}_{\neg\phi \times K}$ contains a self-fulfilling path.*

Theorem 11 *The TKS K validates the TLTL specification ψ if and only if it does not satisfy $\neg\phi$.*

6.4 Complexity of Model Checking

The size of the product tableau $\mathcal{A}_{\neg\phi \times K}$ is bounded by $\mathcal{O}(|K| \times |\mathcal{A}_{\neg\phi}|)$ or $\mathcal{O}(|K| \times 2^{7|\phi|})$, which is linear in the size of the TKS and exponential in the size of the TLTL specification ϕ . Since deciding the presence of a self fulfilling path can always be done in the worst case in time linear on the size of the product graph, we conclude that *the problem if a TLTL-formula $\psi = \forall t_1 \dots t_k. \phi$ holds in a TKS K can be decided in deterministic time linear in the size of the K and exponential in the length of ϕ .*

Following the argument presented for satisfiability checking in Theorem 8, there exists a non deterministic algorithm which checks if $\mathcal{A}_{\neg\phi \times K}$ contains a self-fulfilling path for $\neg\phi$ using $O(|\phi|)$ space. This renders the model checking also in PSPACE. To check the hardness part, we need to reduce the validity problem for TLTL to model checking, which requires defining a TKS K of constant size such that formula ϕ holds *iff* it is valid in K . Towards that, we further assume that the range of static timing variables are restricted to the interval $[0, M] \subseteq \mathbb{N}$, where the value of M can be approximated by the maximum path delay in the Timeout Kripke structure defined below. The *Path delay* for a specific (acyclic) path starting from some initial location and ending at some designated location is the sum of the maximal possible timeout increments or clock delays (replacing open ended timeout increments with arbitrary values) over

⁴All the timeouts with minimum value are incremented on taking the transition (s, s') .

the transitions across the path. Well-known shortest path algorithms [CLRS01, 580-642], *viz.*, Floyd-Warshall algorithm, Dijkstra's algorithm, can be easily be adapted for calculating such maximal path delay over a given TKS. Now, choose $K = \langle 2^{\mathcal{P} \cup [0, M]}, 2^{\mathcal{P} \cup [0, M]}, 2^{\mathcal{P} \cup [0, M]} \times \{0\} \times \{\star\} \times 2^{\mathcal{P} \cup [0, M]} \rangle$ to be the complete graph over all subsets of $\mathcal{P} \cup [0, M]$.

7 Undecidability of Dense TLTL

We relax the time-progress condition and consider an interpretation of TLTL formulas over a dense time domain. We prove the resulting logic to be highly undecidable by reducing a Σ_1^1 -hard problem to its satisfiability problem.

7.1 2-counter Machines

A *nondeterministic 2-counter machine* \mathbb{M} consists of two counters C_1 and C_2 assuming non negative integer values, and a finite sequence of labeled instructions (e.g., labeled by numbers $1, 2, \dots$). Each instruction may either increment or decrement one of the counters, or jump, conditionally upon one of the counters being zero. When the machine \mathbb{M} executes a non-jump instruction, it proceeds non-deterministically to one of two specified instructions. For example, using programming pseudo-code notation, j^{th} instruction may be either of the following, where $i \in \{1, 2\}$:

$$j \quad : \quad C_i := C_i + 1; \text{ goto } l_1 \text{ or } l_2, \quad (11)$$

$$j \quad : \quad C_i := C_i - 1; \text{ goto } l_1 \text{ or } l_2, \quad (12)$$

$$j \quad : \quad \text{if } C_i = 0 \text{ goto } l_1; \text{ else goto } l_2, \quad (13)$$

where l_1 and l_2 are instruction labels. The configurations of such a \mathbb{M} having $n \geq 0$ instructions are represented by triples $\langle i, c, d \rangle$, where $0 \leq i < n$ is the instruction label, and $c \geq 0, d \geq 0$ are the current values of the counters C_1 , and the counter C_2 respectively. The relation between consecutive configurations can be defined in an obvious way. A computation of \mathbb{M} is a ω sequence of related configurations, beginning with the initial configuration, which is usually taken as $\langle 0, 0, 0 \rangle$. Importantly, 2 counter machines are Turing complete [HMU06]. For more details on counter machines see [HMU06, Chap. 8], [Jon97, Chap. 7-8].

The computation of a counter machine is called *recurring* if it contains infinitely many configurations with the value of the instruction counters being 0. It was shown in [AH94] that the problem of deciding if a given nondeterministic 2-counter machine has a recurring computation is Σ_1^1 -hard.

7.2 Dense TLTL

Let us relax the time-progress condition (m_2) and extend the expressive power of TLTL by providing a dense semantics to it, *i.e.*, we assume that between any two given time points there is another time point. We assume our time domain as non-negative rationals $\mathbb{Q}^{\geq 0}$ with dense linear order induced by usual ' $<$ ' relation, which is irreflexive, comparability-permissible and transitive.

The technique we use to prove the undecidability of dense TLTL follows closely the one described in [AH94, Section 4.4] to prove similar result for TPTL.

We need a successor function \mathcal{S} on the underlying time domain $\mathbb{Q}^{\geq 0}$. This function, when applied to an element in $\mathbb{Q}^{\geq 0}$ will return a unique element greater than the original element. \mathcal{S} satisfies the following axioms: i) $q < \mathcal{S}(q)$ for all $q \in \mathbb{Q}^{\geq 0}$ and, ii) $q < q' \Rightarrow \mathcal{S}(q) < \mathcal{S}(q')$ for all $q, q' \in \mathbb{Q}^{\geq 0}$. Note that owing to the denseness of $\mathbb{Q}^{\geq 0}$, arbitrary many time points could be squeezed into a finite interval with the application of successor. For notational convenience, $\mathcal{S}(q)$ will be represented as q^+ in the following discussion.

We encode a computation of \mathbb{M} by using propositions $p_0, p_1, \dots, p_n, r_1$ and r_2 , precisely one of which is true in any state. The configuration $\langle i, c, d \rangle$ of \mathbb{M}

is represented by the finite sequence $p_i, \overbrace{r_1, \dots, r_1}^c, \overbrace{r_2, \dots, r_2}^d$ of states.

The initial configuration $\langle 0, 0, 0 \rangle$ can be encoded using a proposition p_0 . The recurrence condition can be encoded as $(\Box \Diamond p_0)$. It is possible to have the k -th configuration of a computation of \mathbb{M} correspond to the finite sequence of states that is mapped to the interval $[t, t^+)$. We force the time to increase by a strictly positive amount between each successive states using $\Box(x = t \Rightarrow \bigcirc(x > t))$. Now we can copy groups of r -states by establishing a one-to-one correspondence of $r_j (j = 1, 2)$ -states at time t and time t^+ . In the following we assume that t_0, t_1, t_2, \dots , stand for static timing variables.

Let us consider the instruction (11) $j : C_2 := C_2 + 1$; *goto* l_1 or l_2 , which increments the counter C_2 and proceeds nondeterministically to either instruction l_1 or l_2 . We can encode this computation by the following TLTL-formula:

$$\Box(\phi \Rightarrow (\psi_1 \wedge \psi_2 \wedge \psi_3(r_1) \wedge \psi_3(r_2) \wedge \psi_4^{r_2})),$$

where

$$\begin{aligned} \phi : & \quad x = t \wedge p_j \\ \psi_1 : & \quad \Diamond(x = t^+ \wedge (p_{l_1} \vee p_{l_2})) \\ \psi_2 : & \quad \Box(x = t_1 \wedge \bigcirc(x = t_2 \wedge x < t^+) \Rightarrow \Diamond(x = t_1^+ \wedge \bigcirc(x = t_2^+))) \\ \psi_3(r_j) : & \quad \Box((x = t_3 \wedge x < t^+ \wedge r_j) \Rightarrow \Diamond(x = t_3^+ \wedge r_j)) \\ \psi_4^{r_2} : & \quad \Box((x = t_4 \wedge \bigcirc(x = t^+)) \Rightarrow \Diamond(x = t_4^+ \wedge \bigcirc r_2 \wedge \bigcirc \bigcirc (x = t^{++}))) \end{aligned}$$

The formula ϕ specifies that the current state at time t corresponds to instruction j . The first conjunct ψ_1 ensures the proper progression to one of the two specified instructions, l_1 or l_2 at time t^+ . The second conjunct ψ_2 establishes a correspondence between states in successive intervals $[t, t^+)$ and $[t^+, t^{++})$ representing configurations while the formula $\psi_3(r_j)$ copies r_j -states in the corresponding states from first interval to the next. The last conjunct $\psi_4^{r_2}$ adds a r_2 -state at the end of next configuration, as required by the increment operation. In case of counter C_1 getting incremented, we will have $\psi_4^{r_1}$ instead of $\psi_4^{r_2}$ specifying an addition of a r_1 state at the beginning of the r -state sequence in the next configuration:

$$\psi_4^{r_1} : \Box((x = t \wedge \bigcirc(x = t_4 \wedge ((r_1 \vee r_2) \vee (p_{l_1} \vee p_{l_2})))) \Rightarrow \Diamond(x > t^+ \wedge x < t_4^+ \wedge r_1 \wedge \bigcirc(x = t_4^+)))$$

Next, for the instruction (12) $j : C_2 := C_2 - 1$; *goto* l_1 or l_2 , which specifies a decrement operation on C_2 , we copy all r_1 states as specified by $\psi_3(r_1)$ above. However we copy the r_2 states excluding the last copy in the sequence. This is achieved by first modifying ψ_3 for r_2 as follows:

$$\psi_3'(r_2) : \Box((x = t_3 \wedge x < t^+ \wedge r_2 \wedge \neg \bigcirc(x = t^+)) \Rightarrow \Diamond(x = t_3^+ \wedge r_2))$$

and then rewriting $\psi_4^{r_2}$ as

$$\psi_4^{r_2} : \Box((x = t_4 \wedge x < t^+ \wedge r_2 \wedge \bigcirc(x = t^+)) \Rightarrow (x = t_4^+ \wedge \bigcirc(x = t^{++})))$$

In case of decrement on C_1 , we copy all the r_2 states as specified by $\psi_3(r_2)$, however copy the r_1 states only after excluding the first copy in the sequence. This is achieved by modifying ψ_3 for r_1 as follows:

$$\begin{aligned} \psi'_3(r_1) : & \Box(\psi_3^{yes} \wedge \psi_3^{no}), \text{ where} \\ \psi_3^{yes} : & (x = t_3 \wedge x < t^+ \wedge r_1) \Rightarrow \Diamond(x = t_3^+ \wedge r_1) \\ \psi_3^{no} : & \neg(x = t \wedge \bigcirc(x = t_3 \wedge r_1) \Rightarrow \Diamond(x = t_3^+ \wedge r_1)) \end{aligned}$$

Finally, we encode the *if-else* instruction (13) $j : \text{if } C_1 = 0 \text{ goto } l_1; \text{ else goto } l_2$; as following:

$$\Box(\phi \Rightarrow (\psi'_1 \wedge \psi'_2 \wedge \psi'_3(r_1) \wedge \psi'_3(r_2)))$$

where

$$\begin{aligned} \psi'_1 : & ((x = t \wedge \bigcirc(r_2 \vee (p_{l_1} \vee p_{l_2}))) \Rightarrow \Diamond(x = t^+ \wedge p_{l_1})) \\ & \vee ((x = t \wedge \bigcirc(r_1)) \Rightarrow \Diamond(x = t^+ \wedge p_{l_2})) \\ \psi'_2 : & \Box(x = t_1 \wedge \bigcirc((x = t_2 \wedge x < t^+) \Rightarrow \Diamond(x = t_1^+ \wedge \bigcirc(x = t_2^+)))) \\ \psi'_3(r_j) : & \Box((x = t_3 \wedge x < t^+ \wedge r_j) \Rightarrow \Diamond(x = t_3^+ \wedge r_j)) \end{aligned}$$

In case of $j : \text{if } C_2 = 0 \text{ goto } l_1; \text{ else goto } l_2$, we modify ψ'_1 as follows:

$$\psi''_1 : ((x = t \wedge \neg\Diamond(r_2)) \Rightarrow \Diamond(x = t^+ \wedge p_{l_1})) \vee ((x = t \wedge \Diamond(r_2)) \Rightarrow \Diamond(x = t^+ \wedge p_{l_2}))$$

Thus for this 2-counter machine, \mathbb{M} we can construct a formula $\phi_{\mathbb{M}}$ such that $\phi_{\mathbb{M}}$ is satisfiable iff \mathbb{M} has a recurring computation. Hence the satisfiability of TLTL is Σ_1^1 -hard.

We observe that the satisfiability of a TLTL formula ψ can be always expressed as a Σ_1^1 -sentence implying the existence of a model for ψ . Since $\mathbb{Q}^{\geq 0}$ is countable, ψ will also have a countable model. Thus any state sequence σ for ψ can be encoded by finitely many infinite sets of natural numbers in first-order arithmetic; say, one for each proposition p in ψ , characterizing the states in which p holds. It is easy to see ψ , as a first-order predicate holds in σ . We conclude that the satisfiability of TLTL formulas is in Σ_1^1 .

Theorem 12 *The satisfiability problem for dense TLTL formulas is Σ_1^1 -complete.*

8 Discussion

While existing real-time logics e.g., TPTL [AH94] can specify clock based dense time properties, TLTL is more suitable for expressing properties of timeout based real-time models for the given semantic interpretation using timeout dynamics where granularity of time is defined in terms of timeout updates. As discussed in Section 6, the infinite state space models of real-time systems can be model checked over discrete time TLTL using the proposed abstractions on the Kripke structure.

Though we only consider minimum of the timeout values using a dummy variable y , dynamic constraints involving individual timeouts (e.g., constraints of the form $x \leq \tau_j + c$, where $\tau_j \in \mathcal{TO}, c \in \mathbb{N}$) can be easily included in the

vocabulary of the logic because the existing tableaux procedure presented in Section 5.3 can be seamlessly extended using the fact that in any state s it is the case, $\forall \tau_j \in \mathcal{TO}. s(\tau_j) \geq s(y)$. Similarly extending the logic with constraints involving congruences similar to TPTL and arithmetic expressions involving more than one timing variables similar to XCTL would enhance the expressive power of the logic. Digitizability [HMP92] is yet another important property for applying discrete time verification techniques on dense time logics and models. Quite often, not all the formulas in dense time logics are digitizable, thus not amenable to discrete time verification. It remains to be seen which fragment of TLTL is digitizable. We conclude by trying to compare TLTL with Monadic Second Order Logic of Order (MSO). It will be a routine exercise to show that TLTL can be embedded in MSO, following the work [AH93]. However as a future work, it would be interesting to characterize the fragment of MSO, for which TLTL will be expressively complete.

Acknowledgment Both the authors did this work when they were with HTS Research, Bangalore, India.

References

- [AD94] R. Alur and D.L. Dill. A Theory of Timed Automata. *Theoretical computer science*, 126(2):183–235, 1994.
- [AFH96] R. Alur, T. Feder, and T.A. Henzinger. The Benefits of Relaxing Punctuality. *Journal of the ACM (JACM)*, 43(1):116–146, 1996.
- [AH92] R. Alur and T.A. Henzinger. Logics and Models of Real-time: A Survey. In *Proceedings of the Real-Time: Theory in Practice, REX Workshop*, volume 600 of *LNCS*, pages 74–106. Springer, 1992.
- [AH93] R. Alur and T.A. Henzinger. Real-time Logics: Complexity and Expressiveness. *Information and Computation*, 104(1):35–77, 1993.
- [AH94] R. Alur and T.A. Henzinger. A Really Temporal Logic. *Journal of the ACM*, 41(1):181–203, 1994.
- [BCM05] P. Bouyer, F. Chevalier, and N. Markey. On the Expressiveness of TPTL and MTL. In *Proceedings of the 25th international conference on foundations of software technology and theoretical computer science*, volume 3821 of *LNCS*, pages 432–443. Springer-Verlag, 2005.
- [BD98] D. Bosnacki and D. Dams. Integrating Real Time into Spin: A Prototype Implementation. In *Proceedings of the Formal Description Techniques and Protocol Specification, Testing and Verification (FORTE/PSTV)*, pages 423–439. Kluwer, BV, 1998.
- [BDL04] G. Behrmann, A. David, and K.G. Larsen. A Tutorial on UPPAAL. In *4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM-RT’04)*, volume 3185 of *LNCS*, pages 200–236. Springer-Verlag, New York, 2004.

- [Buc60] J.R. Buchi. On a Decision Method in Restricted Second Order Arithmetic. In *Proceedings of International Congress on Logic, Methodology and Philosophy of Science*, pages 1–12. Stanford University Press, 1960.
- [CLRS01] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT press, 2001.
- [dMOR⁺04] L. de Moura, S. Owre, H. Rue, J. Rushby, R. Alur, and D. Peled. SAL 2. In *Proceedings of International Conference on Computer-Aided Verification (CAV'04)*, volume 3114 of *LNCS*, pages 496–500. Springer-Verlag, 2004.
- [DS04] B. Dutertre and M. Sorea. Modeling and Verification of a Fault-Tolerant Real-time Startup Protocol using Calendar Automata. In *Proceedings of FORMATS/FTRTFT*, volume 3253 of *LNCS*, pages 199–214. Springer, 2004.
- [HLP90] D. Harel, O. Lichtenstein, and A. Pnueli. Explicit Clock Temporal Logic. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 402–413. IEEE Computer Society, 1990.
- [HMP92] T.A. Henzinger, Z. Manna, and A. Pnueli. What Good Are Digital Clocks? In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, pages 545–558. Springer-Verlag, 1992.
- [HMU06] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-wesley, 2006.
- [Jon97] N.D. Jones. *Computability and Complexity: From a Programming Perspective*. The MIT Press, 1997.
- [Koy90] R. Koymans. Specifying Real-Time Properties with Metric Temporal Logic. *Real-Time Systems*, 2(4):255–299, 1990.
- [LP85] O. Lichtenstein and A. Pnueli. Checking that Finite State Concurrent Programs Satisfy their Linear Specification. In *Proceedings of the 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, pages 97–107. ACM New York, NY, USA, 1985.
- [OD08] Ernst-Rudiger Olderog and Henning Dierks. *Real-Time Systems: Formal Specification and Automatic Verification*. Cambridge University Press, 2008.
- [Ost89] J.S. Ostroff. *Temporal Logic for Real-time Systems*. Wiley Advanced Software Development Series, 1989.
- [PH88] A. Pnueli and E. Harel. Applications of Temporal Logic to the Specification of Real-time Systems. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 331 of *LNCS*, pages 84–98. Springer, 1988.

- [Pnu77] A. Pnueli. The Temporal Logic of Programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 46–57, 1977.
- [Pra77] V.R. Pratt. Two Easy Theories whose Combination is Hard. Technical report, Massachusetts Institute of Technology, Cambridge, 1977.
- [Sav70] W. J. Savitch. Relationships between Nondeterministic and Deterministic Tape Complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.
- [SC85] AP Sistla and EM Clarke. The Complexity of Propositional Linear Temporal Logics. *Journal of the ACM (JACM)*, 32(3):733–749, 1985.
- [SMR07] I. Saha, J. Misra, and S. Roy. Timeout and Calendar Based Finite State Modeling and Verification of Real-Time Systems. In *Proceedings of 5th International Symposium on Automated Technology for Verification and Analysis (ATVA'07)*, volume 4762 of *LNCS*, pages 284–299. Springer, 2007.
- [SP02] W. Steiner and M. Paulitsch. The Transition from Asynchronous to Synchronous System Operation: An Approach for Distributed Fault-Tolerant System. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, volume 22, pages 329–336. IEEE Computer Society, 2002.
- [TC96] S. Tripakis and C. Courcoubetis. Extending Promela and Spin for Real Time. In *Proceedings of the Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, (TACAS'96)*, volume 1055 of *LNCS*, pages 329–348. Springer Verlag, 1996.